Thermal-Aware Data Flow Analysis

José L. Ayala[1] David Atienza[2] Philip Brisk[2]

[1]DACYA, Complutense University of Madrid – 28040 Madrid (Spain)
E-mail: jayala@fdi.ucm.es
[2]ESL and LAP, EPFL – 1015 Lausanne (Switzerland)
E-mail: {david.atienza, philip.brisk}@epfl.ch

ABSTRACT

This paper suggests that the thermal state of a processor can be approximated using data flow analysis. The results of this analysis can be used to evaluate the efficacy of thermal-aware compilation strategies, or as input to thermal-aware optimizations that occur in the early stages of back-end compilation. We propose different ways how the exploitation of thermal behavior knowledge can be included in the different compilation phases.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: Processor Architectures

General Terms

Algorithms, Theory

Keywords

Thermal Management, Compiler.

1. INTRODUCTION

Thermal management has become an increasingly important issue in modern semiconductor devices. In particular, steep thermal gradients have been shown to significantly reduce the reliability of silicon systems. In processors, one particular area of concern is the *register file (RF)*, which has high power density and is accessed every cycle [1]. In response, several research groups have recently proposed thermal-aware register assignment techniques for the RF [2, 3], or thermal-aware instruction binding in VLIW processors [4]. One of the key challenges for thermal-aware compilation in the near future is to model the effects of program transformations on power density and thermal gradients. State-of-the-art thermal emulation tools require compiled programs in order to characterize the thermal state of the processor [5]; this limits their usage, in practice, to feedback-driven optimization frameworks.

This paper suggests a more radical approach, namely, a compiler may be able to predict, with reasonable accuracy, the thermal state of the processor at every point in the program. Although our idea is to consider the benefits and possibilities of thermal-aware data flow analysis in general, the motivating example and following discussion will emphasize optimizations relating to the RF as an exploratory case study. The key point of the analysis—and the reason that we consider the idea to be "wild and crazy"—is that, as the analysis is applied prior to register allocation and assignment, and at this stage there is no information about the layout of the RF and the placement of registers, some accuracy could be lost due to the taken assumptions; but, this is the only alternative to feedback-driven thermal optimization.

DAC'09, July 26-31, 2009, San Francisco, California, USA

2. MOTIVATING EXAMPLE

Two variables *interfere* in a program if their lifetimes overlap. Interfering variables cannot be assigned to the same register or memory location in order to avoid any overwriting. As a result, when a compiler assigns a register to variable *v*, it can choose from any register *not* assigned to a variable that interferes with *v*. In general, the compiler maintains an ordered list of registers and selects the first one in the list that is free. As the list is always traversed in order, the same small set of registers is chosen again and again. As far as performance is concerned, this is generally permissible, as registers are assumed to be interchangeable; however, repeatedly choosing the same registers for assignment is not the best choice from a thermal management point of view.

Fig. 1 shows thermal maps for three register assignment policies: choosing the first free register from an ordered list (Fig. 1(a)); randomly choosing a free register (Fig. 1(b)); and a "chessboard" pattern (Fig. 1(c)) [2]. The first two policies result in clear RF hot spots with steep thermal gradients; the chessboard policy, however, shows a homogenized temperature map, which occurs because the accesses are distributed uniformly across a large surface. The chessboard policy, however, only works if the program only uses half of the registers in the RF. Indeed, if register pressure is high, then all registers will be used, and may be accessed repeatedly. If certain registers are accessed more than others, then thermal gradients may still appear and reliability can suffer even trying to apply the chessboard pattern of Fig. 1(c).

3. BASICS OF DATA FLOW ANALYSIS

A compiler uses data flow analysis to determine properties about a program that are either required for correctness or useful for optimization. Depending on the needs of the analysis, different types of information must be propagated through the data flow solver. For example, *liveness analysis* asks whether a variable is *live* at each point in the program—if so, then a storage location must be allocated for that variable. In this case, a single bit of information per variable is required. *Bitwidth analysis* [7], which is more complex, propagates an interval for each variable, which represents the range of possible values that it can take; the bitwidth can be derived from this interval. The thermal analysis, proposed in the next section, must propagate a floorplan-aware estimate of the thermal state of the processor, which is considerably more complex than a bit or an interval for each variable. The thermal state is a continuous function that can only be approximated, typically as a discrete set of points. The fidelity of the analysis will depend on the granularity of the approximation—increasing the number of points would increase accuracy, but at the cost of increased computation time.

4. THERMAL DATA FLOW ANALYSIS

The proposed data flow analysis would be a forward analysis [6]. For simplicity, we describe it in the context of a single procedure. A pseudocode description is shown in Fig. 2. The analysis repeatedly computes the thermal state of the RF following each instruction, which relates the technology coefficients of logic activity and peak

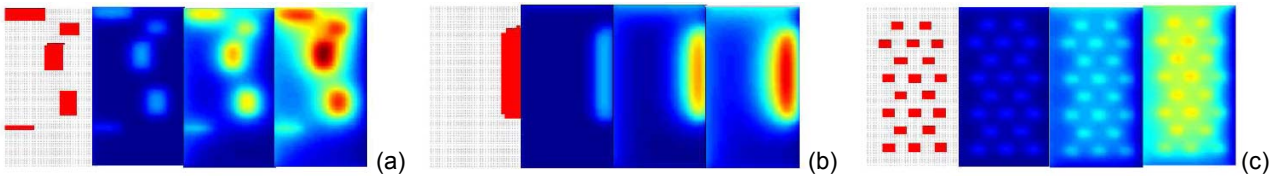

Figure 1. Thermal maps for register assignment policies: (a) deterministic order; (b) random; and (c) chessboard.

power found in the thermal models [1, 5]. The coefficients are linked in an analytical way to the high-level information of instruction execution and variables assignment found in the early compilation stages. If the change in thermal state prediction for at least one instruction exceeds a user-supplied parameter (δ), then another iteration is required; otherwise, the analysis terminates and the thermal state following each instruction is output.

```
Do
    Boolean: stop ← True
    For each basic block B
        For each instruction I∈ B, taken in forward order
            Estimate thermal state after I
            If the change in I's thermal state exceeds δ
                stop ← False
            EndIf
        EndFor
    EndFor
While( stop = False )
Output the thermal state of each instruction
```

Figure 2. Pseudocode of proposed data flow analysis.

Unlike traditional data flow analyses, in the proposed thermal-aware analysis does not appear to be a way to guarantee convergence; however, if the analysis does not converge after a reasonable number of iterations ("reasonable" must be determined empirically or user-defined), this suggests that the thermal state of the program may be too difficult to predict at compile time due to a very irregular data usage. Thus, to ensure reliability, the program could be re-optimized so that its thermal state becomes more predictable. Also, the result of the analysis phase can be used to conduct the compilation process achieving a temperature-aware compilation at different stages.

The proposed thermal analysis makes the most sense if applied after register assignment, as the precise registers that are accessed by each instruction are known, but then no aggressive thermal-aware high-level optimizations are possible. However, the more ambitious possibility that we propose in this paper, which has never been considered before, would be to develop predictive analyses that would be performed at earlier stages of compilation, i.e., before register allocation and assignment, or perhaps even before instruction scheduling. While it is not easy to promote the thermal information to the early compilation stages, the development of a set of rules that qualify the impact of the compiler decisions on the thermal profile will allow the envisioning of later thermal-aware compilation without the feedback of temperature information, which involves a time-consuming thermal simulation phase of the target processor [2, 5].

The goal of these analyses would not be to accurately predict the thermal state of the process at each point; instead, the goal would be to determine precisely which parts of the program are likely to exacerbate power density and thermal problems in the RFs, and to determine which variables are most likely to be involved. In particular, if just two variables are involved, they can easily be assigned to registers in disparate regions of the RF; however, when more variables are likely to create hot spots, it becomes increasingly difficult to assign them to registers in different regions of the RF, especially when register pressure is high.

For the purposes of thermal management, the greatest benefit will be achieved by spilling these "critical" variables to memory, or splitting them (via copy insertion) to spread their accesses across a multitude of registers. As it has been seen in the presented thermal maps, spreading (in space) the accessed registers reduces the appearance of hotspots and thermal gradients. Also, the thermal diffusion between the accessed registers homogenizes the temperature on the device and improves its reliability by decreasing leakage. However, power reduction techniques based on switching off register banks could not theoretically be applied after the spread register assignment, and a compromise between these types of techniques for different optimization metrics can be explored at the compiler level.

Other possible optimizations that could be driven by this analysis refer to spreading accesses to registers in time, either using instruction scheduling, to avoid consecutive accesses to already hot registers, or using register promotion (i.e., promoting some memory-resident variables into registers), which would help on avoiding the thermal gradients between hot and cold registers, by making more uniform the use of registers in time. Finally, the insertion of NOP instructions gives the RF a chance to cool down between accesses in extremely hot situations, although it can affect overall system performance and should be applied only if no other option to cool down the system is feasible.

5. CONCLUSIONS

This paper proposes that compilers can estimate the thermal state of a processor in early stages of compilation using data flow analysis; in particular, we have suggested how different compiler transformations can guide thermal-aware optimization methods for the RF, which is the first step to develop even more general thermal-aware high-level optimization in the future. In the long-term, our goal is to develop comprehensive data flow thermal analyses and rules relating to all parts of the processor and to use them to better understand the impact of high-level transformations on thermal gradients and power density at the same time.

REFERENCES

[1] Srinivasan, J., et al. Predictive dynamic thermal management for multimedia applications. In *Proc. 17th ICS*, pp. 109-120, 2003.

[2] Atienza, D., et al. Reliability-aware design for nanometer-scale devices. In *Proc. ASPDAC*, pp. 549-554, 2008.

[3] Zhou, X., et al. Compiler-driven register re-assignment for register file power-density and temperature reduction. In *Proc. DAC*, pp. 750-753, 2008.

[4] Schafer, B. C., et al. Temperature-Aware Compilation for VLIW Processors. In *Proc. RTCSA*, pp. 426-431, 2007.

[5] Atienza, D., et al. HW-SW emulation framework for temperature-aware design in MPSoCs. *ACM TODAES*, 12(3), pp. 1-26, August, 2007.

[6] Cooper, K. D., and Torczon, L. Engineering a Compiler, *Morgan-Kaufmann*, San Francisco, CA, USA, 2003.

[7] Stephenson, M., et al. Bitwidth analysis with application to silicon compilation, In *Proc. of PLDI*, pp. 108-120, 2000.